UNIVERSITY OF CALIFORNIA, DAVIS
DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING

COURSE: WATER RESOURCES SIMULATION (ECI 146)

INSTRUCTOR: ***Fabián A. Bombardelli***
(fabianbombardelli2@gmail.com, bmbrdll@yahoo.com, fabombardelli@ucdavis.edu)
OFFICE: 3105, Ghausi Hall (former Engineering III building)
Class: Tuesdays and Thursdays-10:30 to 11:50 PM (119 Wellman)
TEACHING ASSISTANTS AND READER: Ms. Jenny Wang, Mr. Arturo Palomino

---

COMPUTER PROBLEM 1: SOLUTION

Problem 1

1-2) What is the physical meaning of smooth and fully-rough regimes?

These are concepts you should recall from ECI141 or any equivalent Fluid Mechanics course. When the regime is turbulent smooth, the viscous sub-layer is much larger than the roughness height. Thus, the viscous sub-layer covers completely the roughness height and beyond. In the fully-rough regime, the viscous sub-layer is much smaller than the roughness height, i.e., the latter protrudes the viscous sub-layer. In this region, the friction factor only depends on the relative roughness of the pipe.

1-3) Please discuss the difference between the "true" and "approximate" errors in the bisection method. Why do we use the latter if these two errors are different?

True error is the absolute value of the difference between the exact root and the guessed root, divided by the former. Approximate error is the absolute value of difference between the last guess and the previous one, divided by the former. Since we do not know the exact root (because it would imply knowing it with infinite number of digits), we do not know the true error, so we have to use the approximate error as the stopping criterion in iterative algorithms. Since the approximate error is larger than the true error, we are safe in using the former value of the error.

1-4) Which method is more robust? Which method is more accurate?

By the word "robust," we refer to a method which provides a solution regardless of the function of which we would like to find a root.

Among the three methods used in the problem below, bisection is a robust method. It always converges to the solution regardless of how the function behaves. Obviously, there is a price for this desirable capability: The bisection method is not as fast as Newton-Raphson or Iteration-of-a-point (Fixed-point) methods (there are exceptions,

but roughly speaking bisection is slower than the other alternatives). The initial guess, inherent nature of the solution, and search interval are crucial factors and may change the situation.

The accuracy of all methods is associated with the desired error tolerance. Therefore, if the error tolerance is set identically, ***all*** iterative algorithms will achieve the same accuracy.

Problem 2

a) Sample Programs in MATLAB

```
%%
% University of California, Davis
% Department of Civil and Environmental Engineering
% ECI 146
% Instructor: Prof. Fabian Bombardelli
% Sample program for Newton-Raphson method
% This program finds the root of "5x^2-4=0" with Newton
% Raphson method
% Code by: Kaveh Zamani
%% initialization
clear all
% clears all the current variables from the memory
clc
% clears the command window
tolerance = input('Stopping criteria tolerance ?')
x_initial = 0.5;
x_old= x_initial;
error = 1;
% one is assigned to error to enter the below while loop
iteration_number = 0;
%% Newton-Raphson While loop
while (error > tolerance)
    f_x = 5*x_old^2 - 4;
    f_prim_x = 10*x_old;
    x_new = x_old - f_x/f_prim_x;
    error = abs((x_old - x_new)/x_new)*100;
    iteration_number = iteration_number + 1;
    % the variables/expressions without semicolon at the end are
printed on the screen
    iteration_number
    x_old = x_new
end

disp('Newton_Raphson method has converged!')
X = x_new
```

```
% University of California, Davis
% Department of Civil and Environmental Engineering
% ECI 146
% Instructor: Prof. Fabian Bombardelli
% by Kaveh Zamani
% This program finds the root of "x=cos(x)" between 0 and 1 by the
% method of False Position
% method is also called "Regula Falsi" method.
%% initialization
clear all
% clears all the current variables from the memory
clc
% clears the command window
tolerance = 0.000001;
max_num_iteration = 10000;
x_left_initial  = 0;
x_right_initial = 1;
x_left = x_left_initial;
x_right= x_right_initial;
x_old = 2;

%% False-Position loop
for i=1:max_num_iteration;

    f_left = x_left - cos(x_left);
    f_right = x_right - cos(x_right);
    x_mid = - f_right*(x_right - x_left)/(f_right - f_left)  + x_right;
    f_mid = x_mid - cos(x_mid);
    x_new = x_mid;

    error = abs((x_new-x_old)/x_old)*100;

    if(error < tolerance)
        disp('False Position method has converged!');
        X = x_mid
        break
        % break out of if condition
    elseif ( f_right*f_mid > 0)
        x_right = x_mid;
    else
        x_left = x_mid;
    end
    iteration_number = iteration_number + 1;
% the variables/expressions without semicolon at the end are printed on
the screen
    x_old = x_new;

end
```

```
%
% University of California, Davis
% Department of Civil and Environmental Engineering
% ECI 146 Water Resource Simulation
% Instructor: Prof. Fabian Bombardelli
% Sample program for bisection method
% by Kaveh Zamani
% This program finds the root of "x^2-2x=0" between 1 and 4 by
Bisection method
%
%% initialization
clear all
% clears all the current variables from the memory
clc
% clears the command window
tolerance = 0.000001;
max_num_iteration = 10000;
x_left_initial  = 1;
x_right_initial = 4;
x_left = x_left_initial;
x_right= x_right_initial;

%% Bisection loop
for i=1:max_num_iteration;
    x_mid_new = (x_left + x_right)/2;
    f_left = x_left^2 - 2*x_left;
    f_right = x_right^2 - 2*x_right;
    f_mid = x_mid_new^2 - 2*x_mid_new;

    error = abs((x_mid_new - x_mid_old)/x_mid_old)*100;

    if(error < tolerance)
        disp('Bisection method has converged!');
        X = x_mid_new
        break
        % break out of if condition
    elseif ( f_right*f_mid > 0)
        x_right = x_mid_new;
    else
        x_left = x_mid_new;
    end
    iteration_number = i
    % the variables/expressions without semicolon at the end are
printed on the screen
    x_mid_old = x_mid_new;
end
```

b) Sample program written in Fortran for bisection with using the function as part of the stopping criterion:

```
      !Calculation of the friction factor using
      !Colebrook-White's formula

      !Enter the data

            write(*,*) 'Enter roughness Ratio'
            read(*,*) ED
            write(*,*) 'Enter Reynolds number'
            read(*,*) RE

      !Interval

            write(*,*) 'Enter minimum value'
            read(*,*) FMIN
            write(*,*) 'Enter maximum value'
            read(*,*) FMAX

      !Iteration procedure

            iter=0
10          iter=iter+1
            FR=0.5*(FMIN+FMAX)
            F=1.+sqrt(FR)*0.869*log(ED/3.7+2.51/(RE*sqrt(FR)))
            if (F.gt.0.) FMIN=FR
            if(F.lt.0.) FMAX=FR
            if (abs(F).gt.1.e-6) go to 10

      !output: friction factor and number of iteration
            write (*,*) 'the friction factor is', FR
            write(*,*) iter
            end
```

The above FORTRAN code follows the flow chart of the notebook.

c) Results:

Bisection Method's Number of Iterations for Different Tolerances (Relative Error)							
Number	e/D	Re	*F*	# of iterations			Region
				10^{-2}	10^{-3}	10^{-4}	
1	0.0008	3.0×10^{6}	0.01872	16	19	22	Fully rough/Transition
2	0.00005	3.0×10^{6}	0.01145	17	19	23	Transition
3	0.00001	3.0×10^{7}	0.00843	17	20	23	Transition
4	0.002	3.0×10^{7}	0.02341	16	18	22	Fully rough
5	0.015	3.0×10^{7}	0.04364	15	17	21	Fully rough
6	10^{-10}	3.0×10^{5}	0.01445	17	19	22	Smooth
7	0.002	3.0×10^{5}	0.02400	16	18	21	Transition
8	0.03	3.0×10^{5}	0.05722	15	17	20	Fully rough
9	0.002	3.0×10^{4}	0.02807	16	18	21	Transition
10	0.01	3.0×10^{4}	0.03979	15	17	21	Transition
11	10^{-10}	3.0×10^{2}	0.21333	No iteration $f=\frac{64}{Re}$			Laminar

NOTE: The number of iterations depends on initial interval and tolerance level. (Even compiler settings may change it.)

We can see the effect of the tolerance on the number of iterations. Since the tolerance is inversely proportional to the accuracy, when we increase the required accuracy (by decreasing the tolerance), we increase the number of iterations too, as shown in the table above.

Final remarks:

a) Please keep in mind that we are controlling the accuracy through the tolerance; the issue here is how fast we obtain the solution.
b) 15 to 20 iterations are not too fast to obtain a root. (Imagine that you need to perform this computation thousands of times.) This is why this is a "beginning method." The best solution is to combine this with a "refining" method.
c) How fast a method goes to the root is called "convergence," while whether a method is capable of providing solutions or not for any given function is called "robustness."

Problem 3

- Bernoulli equation: $\frac{P_1}{\rho g}+z_1+\frac{V_1^2}{2g}=\frac{P_2}{\rho g}+z_2+\frac{V_2^2}{2g}+h_f$

$P_1=P_2=P_{atm}$ and $V_1=V_2=0$, so $h_f=\Delta z$

$h_f=f\frac{L}{D}\frac{V^2}{2g}$ => $8=f\frac{100}{0.3}\frac{V^2}{2\times9.81}$ (I) => two unknowns and one equation

$$V = \left(\sqrt{\frac{2gDh_f}{L}}\right)\frac{1}{\sqrt{f}}$$

- Colebrook–White equation:

$\frac{1}{\sqrt{f}} = -2\log_{10}\left(\frac{\varepsilon/D}{3.7} + \frac{2.51}{Re\sqrt{f}}\right) = -2\log_{10}\left(\frac{0.0002}{3.7} + \frac{2.51}{\frac{0.3\,V}{2\times10^{-5}}\sqrt{f}}\right)$ (II) => two unknowns and two equations

=> (I) and (II) system of equations for V and f
Starting with f=0.002 and V = 5 m/s we have:

f= 0.0201 and V=4.841 m/s

The same result can be obtained by directly replacing (I) in (II).

Note

If you want to check your calculations online, you can insert your equation in http://www.wolframalpha.com/ It is a solver by Wolfram Mathematica.

Or you can use MATLAB for numerical solution of implicit equations, just type:
x=solve ('x^2-x*sin(x)-3=0') and hit Enter key

x = -2.187372072354580021409197361238 6